THE STRUCTURE OF THE COMPUTER UTILITY

Professor J. B. Dennis

Project MAC,
Massachusetts Institute of Technology,
Cambridge,
Massachusetts, 02139,
U.S.A.

Abstract:

Some of the trends in the design of future systems are outlined and the ways in which these developments will effect computer systems and the challenges which will be presented to future systems designs are mentioned.

Abstractions of computer systems are used to formulate mathematical models from practical computer systems problems. The effect of these trends in systems design on computer science education are also discussed.

Rapporteurs:

Mr. C. R. Snow
Mr. L. B. Wilson

attention has been given to them. In these arguments two assumptions are generally made, firstly that in general purpose computer systems there will always be a hierarchy of physical storage media and not infinite core memory, and secondly programming generality is required. It is asserted that, in such a system, the storage allocation decisions must be made by the system itself as no procedure can be expected to know the storage requirements of any other procedure it may use. In the machine representation of a procedure, an identifier of information must not imply where that information is stored; hence there is a need for location independent addressing or virtual memory.

The next problem is the criterion by which the system moves information. This must be 'upwards' on demand because the system cannot anticipate the need of procedures for specific references. By 'upwards' is meant towards levels having smaller acces times.

Further to this it is necessary to determine the unit of information which is to be moved upward on demand and the page size required. Up to now large sizes of page have been used due to limitations in the hardware; however, a large page size is inconvenient as a whole page is brought into core when a particular word is demanded but the rest of the page may be of little use. Thus, large quantities of useless information may be loaded and also the channels between core and drum may get congested. Programmers have tended to circumvent this difficulty by putting related information on the same page, but this runs counter to the idea of programming generality. Hence small page sizes seem desirable and we can see, in the IBM 360 Model 85, a trend in this direction. Professor Dennis believed that the next major achievement in commercial computer systems would be a small page size and the replacement of software paging techniques by hardware. As a smaller page size will produce a lower bound on program running time with sequential operation, it is the smaller page size which will lead to an increase in parallel computation.

2. Computer System Architecture

In this section Professor Dennis described some ideas which might be useful for some of the problems in computer system architecture, the two main ideas presented being a model for information structures and the concept of the application of procedures in a computer system.

1. Classification of Information Systems

The information systems considered below all include the concept of 'time sharing' either in its original sense of the multiplexing of hardware amonst several activities or in its more modern sense of man/computer interactive terminal systems.

--- Transaction Systems

This type of system, of which the SABRE airline reservation system is one example, is characterized by conventional multi-programming techniques, by having been written in machine code, by not using the concept of virtual memory, by the ability to service requests from many remote terminals and by the whole system being under the control of a corporate entity (e.g. the airline).

--- Dedicated Information Systems

This next type of system, which uses much the same technology as the Transaction System, is characterized by offering some kind of information service to clients outside the organization running the service, (for example a credit bureau service or stock quotation service).

The systems which are man/computer interactive in the development of programs were considered by the speaker to be of a different class to the two types above.

--- Dedicated Interactive System

This type of information system, provides a fixed language service to a large set of users, the best known example of which is perhaps JOSS (Johniac Open Shop System).

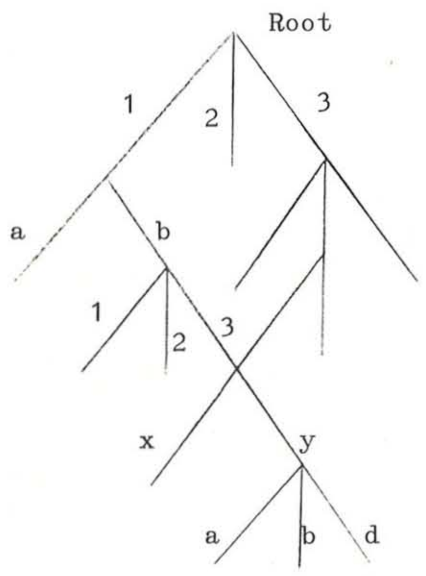
--- General Purpose Interactive Systems

In this type of system a user may write programs in several high-level languages and perhaps an assembly language as well. These programs can be edited, tested and run from a remote terminal. An example of such a system is the Dartmouth System (G.E.265).

--- Extensible Systems

An extensible system is a system in which the set of languages available to the user may be extended by the user himself sitting at a remote terminal. The first such system was the Compatible Time-Sharing

An information structure may be represented by a directed graph in which each node may be reached by a directed path from a particular node called the root. Under each node the branches are labelled by unique identifiers. This means that no two branches descending from and node may have the same identifier although the same identifier may be used elsewhere in the structure. Some nodes may be reached by more than one path hence allowing shared sub-structures; however, no directed cycles are allowed. Any node defines a new information structure, namely those nodes and branches that can be reached from that node. Matrices, arrays, lists, etc. are all information structures and it is useful to consider every object in a computer system as an information structure.



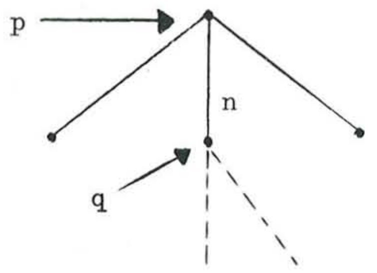
The Universe is an information structure containing representations of all the objects in the system. Any information structure in the system can be considered as a sub-structure of this universe. With the leaf nodes of the structure are associated values which have conventional data types such as real, integer, truth value, string and so on.

A procedure may be thought of as defining a function whose domain is a class of information structures, and whose range is also a class of information structures in that both the input to and the output from a procedure are information structures, even if the procedure has more than one argument.

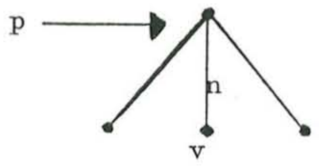
The system may be considered as containing the primitive operations on the basic data types, such as the arithmetic and logical operators, and also a set of operations on the structures themselves. To permit this, a new data type is introduced, the type 'pointer'. The value of a data item of type pointer selects a unique node in the universe.

The operations on structures are:

The select operation which has two arguments, a pointer p and an identifier n. The pointer p defines a structure and the identifier n defines one of the components or branches from the point defined by p. The result of the select operation is a pointer q which is the pointer defining the structure at the node joined to the branch n.



The fetch operation which also has two arguments p and n; however, the component n at the node p must be a leaf. The result of a fetch operation is the value of the data item held at the point defined by p and n.



The assign operation which is the reverse operation to 'fetch'. It has three arguments, p, n and v, and its function is to place the value v at the leaf defined by the pointer p and the identifier n.

So far these operations do not modify the information structure, i.e. the graph stays the same. The procedures are used to get information about the structure or to modify the values at the leaves. It would be more useful to allow these functions to modify the structure itself and implicitly to create new sub-structures in the universe.

The next primitive operation is the delete operation which has two arguments p and n, and its effect is to delete the branch n below the node p. If by this operation, a structure becomes disconnected from the main structure, it is removed from the structure completely. Because of multiple paths, however, it may be that the structure would not completely disappear but only part of it would be erased.

The final primitive, the link operation, has three arguments p, n and q, and has the effect of inserting a branch in between the points defined by the pointers p and q such that q is not below p. This operation must not be allowed to take place if a loop would be formed, but exactly how the loop is to be prevented is not yet clear.

A procedure, made up of these primitive operations, can only get at information 'downwards'; thus, if the procedure is allowed to operate on a structure, it can only affect data items within that structure and only elements within the structure can be altered by the procedure. This is the same as saying that all data items affected by the procedure must be passed as arguments of the procedure, that is, there must be no side-effects.

It is necessary now to discover what limitations must be imposed on programs so that the use of these programs concurrently by several processes does not lead to non-determinacy. A procedure is a partial ordering of operations, where these operations may be any of the primitive operations of the application of another procedure. The conditions under which a procedure, defined in this way, will lead to a deterministic computation must be established for these structure operations, as the answer is already known for procedures which do not involve them. Any number of processes may concurrently read a structure, but if any process is modifying the structure, then all other processes must be denied all access to that structure. The conjecture is that a theory may be developed from this which shows that this is a necessary and sufficient condition for a computation to be deterministic.

This presents rather an attractive view of a procedure, since it gives a general model of an information structure and it includes a way of representing the sharing of information. The suggestion was made that one might take this model and use it to examine implementations in terms of computer systems, and also to assist in the formal definition of programming languages as, in fact, the Vienna group are already doing for PL/1. They have used the class of objects which is essentially the same as the information structure, without sharing, formally to define the language PL/1, and have used it both as a representation of the program in abstract form and also as a representation of the states of an abstract machine which defines the actual execution of the program.

place on programming languages. 'Programs must also bring in the parallelism we need in the systems.'

However, if processor time is going to be negligible with respect to I/O and parallelism in the system is mainly a function of the processor Professor Arden wondered whether parallelism would really gain us very much. Professor Dennis believed that: 'We must increase parallelism in handling I/O requests as well so we can get a more than linear increase of speed here. This may make core memory obsolete because it can only handle single requests which would not be true of semi-conductor memories.'

2. Computer System Architecture

Professor Piloty pointed out that, the Vienna Group had not only worked on PL/1, but that the first part of their work was general for any programming language and they had used it to define ALGOL 60. Professor Dennis agreed but pointed out that what they had not done was to use an abstract representation of a program which could provide a common semantic base for two different programming languages. 'If you look at their abstract machine for defining PL/1, it is not the same as that used to define Algol. What would be interesting is to use their techniques to reduce two programming languages or more to a common semantic base. The model for information structures may be a key to doing this.'

Professor Wedekind asked whether Professor Dennis would consider generalising the 'select downwards' operation concept to include a 'select upwards' as well; to which Professor Dennis replied that he would not want to provide a select upwards operation, because this would allow a procedure to work outside the context provided for it.

Professor Piloty pointed out that, from the set theoretic point of view, 'Universe' was not such a good name to use. Professor Dennis said that he had earlier used the word 'Environment', but found that it had already been used in a more restricted sense. "'Universe' is, I agree, not used in the set theory sense but at present I cannot think of a better word."

3. Some Educational Issues

In reply to a question from Professor Piloty, Professor Dennis said that he did not think that algebra was *per se* especially important to work in computer systems. 'Whilst a computer scientist will need some algebra for example the notion of a semi-group, properties of integers, etc. - a deep treatment of algebra seems unnecessary.' Professor Piloty pointed out, however, that: 'In algebra you are defining operations and giving it structure in much the same way as your abstraction of an information system with its primitive operations', to which Professor Dennis replied that, 'In the development of abstractions, the required mathematics is deduced from the problem rather than forcing the problem into algebra. This latter approach is one I deplore, I have not found modern algebra very useful in computer system work. The only part I can think of is distributive lattice theory which is not generally taught.'

Dr. Ollongren asked the speaker whether he thought that the approach used by Knuth in his first book on the abstract and concrete algorithm was valuable. Professor Dennis agreed that it was very valuable but felt that Knuth took the Von Neuman architecture much more for granted than he was prepared to do.

Dr. Needham noted that the speaker had started by saying that computer science was an engineering subject, but what he taught seemed very abstract. Professor Dennis replied that: 'Although one may object that we are going too fast with this, at some stage it is better to give students the abstractions you have found useful rather than let them find abstractions for themselves through practical programming.' Professor Dennis also mentioned that, apart from the formal teaching, his students had unusual opportunities to get informal experience, either from projects at M.I.T., with consulting houses, vacation jobs and, of course, at high school.

Professor Page noted that: 'In many universities one finds applied mathematicians are purer than the pure mathematicians. This happens when they get very far from the practical problems and get abstractions of abstractions', and wondered whether this danger was also here as one got more into abstractions. Professor Dennis saw this as a danger. 'Some groups have become concerned with their own abstractions and become ingrown. This is not the same as making abstractions of practical systems. Also students in computer science are motivated to do and build things rather than abstractions.' Professor Page asked whether this was also true of professors, to which Professor Dennis replied that:

> 'the danger is greater with them, particularly as the more practical they are the more likely they are to be snapped up by industry!'

Professor Randell asked whether more emphasis was given to teaching why rather than how, to which Professor Dennis replied:

> 'Yes, but they need to know the how otherwise they will not appreciate the why.'

Dr. Browning asked the speaker whether he saw the software houses taking over a lot of the system programming. Professor Dennis replied that, in his view, this would not happen for experimental systems. 'These are much too difficult to specify. If you can make a precise specification then you can hand it out to a software house.'

Professor Dennis was asked by several people about how he taught some of the ideas he introduced in his talk.

Professor Dennis:

'There are several ways to teach these ideas. You can compare the features of several programming languages in order to understand their meaning and relations, then show how the features are obtained by the standard Von Neumann architecture and, where this is unsatisfactory, how this architecture can be extended. However, at M.I.T. a more abstract approach has been used, teaching the notion of algorithm in terms of the abstraction of λ - calculus. A later course develops an abstract model for representing computation for both hardware and program. We use this model to describe computer systems. The ideas of parallelism can be introduced through Dijkstra semaphors and primitives (see his paper on co-operating sequential processes*) and this leads to representing procedures in pure form and also some practical ideas on operating systems.'

We have found λ - calculus perhaps a bit too abstract but we are unlikely to let it go completely. We want to teach programming languages instead of from the point of view of doing a computation, so that stüdents understand the reasons for features in the language and how meaning is given to them - for example, the notions of iteration, recursion, the communication of parameters to a procedure. We are not educating people to be programmers."

* Dijkstra, E. W. (1965): 'Co-operating Sequential Processes'. Report EWD 123, Mathematical Department, Technological University, Eindhoven. (Reprinted in F. Genuys (ed.), 'Programming Languages', 1968, London, Academic Press.)